Apple Assembly Line

$1.50

Volume 4 -- Issue 3 December, 1983

In This Issue...

Demise of Bailey's DataPhile Digest

Unfortunately, we no sooner sent out last month's AAL than we received a letter from the Baileys saying that they have ceased to publish the DataPhile Digest.

Quarterly Disk 13

QD 13 is now ready, and it includes both installments of ProDOS commented source code as listed last month and this. The code is in the format used by the S-C Macro Assembler. (Since the disk also includes the CONVERT S-C TO TEXT program in this issue, all of you can use it!) Quarterly Disks are $15 each, or $45 for a year's subscription.

Subscription Rates

Remember, subscriptions to Apple Assembly Line will be increasing to $18/year effective January 1. Since some of you may not receive this issue (or your renewal notice) until after that date, we'll extend the deadline to January 15 for renewals.

Commented Listing of ProDOS $F90C-F995, $FD00-FE9A, $FEBE-FFFF

............Bob Sander-Cederlof

Last month I printed the commented listing of the disk reading subroutines. This month's selection covers disk writing, track positioning, and interrupt handling. Together the two articles cover all the code between $F800 and $FFFF.

Several callers have wondered if this is all there is to ProDOS. No! It is only a small piece. In my opinion, this is the place to start in understanding ProDOS's features: A faster way of getting information to and from standard floppies. But remember that ProDOS also supports the ProFILE hard disk, and a RAM disk in the extended Apple //e memory.

Further, ProDOS has a file structure exactly like Apple /// SOS, with a hierarchical directory and file sizes up to 16 megabytes.

Further, ProDOS includes support for a clock/calendar card, 80-columns with Smarterm or //e, and interrupts.

ProDOS uses or reserves all but 255 bytes of the 16384 bytes in the language card area (both $D000-DFFF banks and all $E000-FFFF). The 255 bytes not reserved are from $D001 through $D0FF in one of the $D000 banks. The byte at $D000 is reserved, because ProDOS uses it to distinguish which $D000 bank is switched on when an interrupt occurs. The space at $BF00-BFFF is used by ProDOS for system linkages and variables (called the System Global Page).

In addition, if you are using Applesoft, ProDOS uses memory from $9600-BEFF. This space does not include any file buffers. When you OPEN files, buffers are allocated as needed. CLOSEing automatically de-allocates buffers. Each buffer is 1024 bytes long. As you can see, with ProDOS in place your Applesoft program has less room than ever.

Track Seeking: $F90C-F995

The SEEK.TRACK subroutine begins at $F90C. The very first instruction multiplies the track number by two, converting ProDOS logical track number to a physical track number. If you want to access a "half-track" position, you could either store a NOP opcode at $F90C, or enter the subroutine at $F90D.

A table is maintained of the current track position for each of up to 12 drives. I call it the OLD.TRACK.TABLE. The subroutine GET.SSSD.IN.X forms an index into OLD.TRACK.TABLE from slot# * 2 + drive#. There are no entries in the table for drives in slots 0 or 1, which is fine with me. ProDOS uses these slots as pseudo slots for the RAM-based pseudo-disk and for ProFILE, if I remember correctly.

The code in SEEK.TRACK.ABSOLUTE is similar but not identical to code in DOS 3.3. The differences do not seem to be significant.

```
S-C Macro Assembler Version 1.0.....................................$80.00
S-C Macro Assembler Version 1.1 Update..............................$12.50
Full Screen Editor for S-C Macro Assembler..........(reg. $49.00)  $40.00**
    Includes complete source code.
S-C Cross Reference Utility.........................................$20.00
S-C Cross Reference Utility with Complete Source Code...............$50.00
DISASM Dis-Assembler (RAK-Ware).....................................$30.00
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
The Visible Computer: 6502 (Software Masters).......(reg. $50.00)  $40.00**

S-C Word Processor (the one we use!)................................$50.00
    With fully commented source code.
Applesoft Source Code on Disk.......................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.
ES-CAPE:  Extended S-C Applesoft Program Editor.....(reg. $60.00)  $40.00**

AAL Quarterly Disks............................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983

Double Precision Floating Point for Applesoft.......................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Amper-Magic Volume 2 (Anthro-Digital)...............(reg. $35.00)  $30.00
Routine Machine (Southwestern Data Systems).........(reg. $64.95)  $60.00
FLASH! Integer BASIC Compiler (Laumer Research).....(reg. $79.00)  $50.00**
Fontrix (Data Transforms)..........................................  $75.00
Aztec C Compiler System (Manx Software)............(reg. $199.00) $180.00

Blank Diskettes..................................package of 20 for $45.00
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold one disk each................10 for $6.00
Diskette Mailing Protectors.........................10-99:  40 cents each
                                               100 or more:  25 cents each
ZIF Game Socket Extender............................................$20.00
Shift-Key Modifier..................................................$15.00

Grappler+ Printer Interface (Orange Micro).............($175.00)  $150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....($175.00)  $150.00
Buffered Grappler+ NEW!!  Interface and 16K Buffer.....($239.00)  $200.00

Books, Books, Books...........................compare our discount prices!
    "The Apple ][ Circuit Description", Gayler...........($22.95)  $21.00
    "Enhancing Your Apple II, vol. 1", Lancaster.........($17.95)  $17.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "What's Where Guide" (updates first edition)..........($9.95)   $9.00
    "6502 Assembly Language Programming", Leventhal......($18.95)  $18.00
    "6502 Subroutines", Leventhal........................($17.95)  $17.00
  Add $1.50 per book for US postage.  Foreign orders add postage needed.

Whatever Else You Need..........................Call for Our Low Prices

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                  (214) 324-2050                   ***
        *** We accept Master Card, VISA and American Express ***

(** Special price to subscribers only through December 31, 1983.)
 ** Last Chance!!  Order now and save.
```

Disk Writing: $FD00-FE9A

The overall process of writing a sector is handled by code in RWTS, which was listed last month. After the desired track is found, RWTS calls PRE.NYBBLE to build a block of 86 bytes containing the low-order two bits from each byte in the caller's buffer. PRE.NYBBLE also stores a number of buffer addresses and slot*16 values inside the WRITE.SECTOR subroutine. Next RWTS calls READ.ADDRESS to find the sector, and then WRITE.SECTOR to put the data out.

WRITE.SECTOR is the real workhorse. And it is very critically timed. Once the write head in your drive is enabled, every machine cycle is closely counted until the last byte is written. First, five sync bytes are written (ten bits each, 1111111100). These are written by putting $FF in the write register at 40 cycle intervals. Following the sync bytes W.S writes a data header of D5 AA AD.

Second, the 86-byte block which PRE.NYBBLE built is written, followed by the coded form of the rest of your buffer. WRITE.SECTOR picks up bytes directly from your buffer, keeps a running checksum, encodes the high-order six bits into an 8-bit value, and writes it on the disk...one byte every 32 cycles, exactly. Since your buffer can be any arbitrary place in memory, and since the 6502 adds cycles for indexed instructions that cross page boundaries, WRITE.SECTOR splits the buffer in parts before and after a page boundary. All the overhead for the split is handled in PRE.NYBBLE, before the timed operations begin.

Finally, the checksum and a data trailer of DE AA EB FF are written.

Empty Space: $FEBE-FF9A

This space had no code in it. Nearly a whole page here.

Interrupt & RESET Handling: $FF9B-FFFF

If the RAM card is switched on when an interrupt or RESET occurs, the vectors at $FFFA-FFFF will be those ProDOS installed rather than the ones permanently coded in ROM. It turns out the non-maskable interrupt (NMI) is still vectored down into page 3. But the more interesting IRQ interrupt is now vectored to code at $FF9B inside ProDOS.

The ProDOS IRQ handler performs two functions beyond those built-in to the monitor ROM. First, the contents of location $45 are saved so that the monitor can safely clobber it. Second, a flag is set indicating which $D000 bank is currently switched on, so that it can be restored after the interrupt handler is finished. (The second step is omitted if the interrupt was caused by a BRK opcode.)

If the IRQ was not due to a BRK opcode, a fake "RTI" vector is pushed on the stack. This consists of a return address of $BF50 and a status of $04. The status keeps IRQ interrupts disabled, and $BF50 is a short routine which turns the ProDOS memory back on and jumps up to INT.SPLICE at $FFD8:

```
       BF50-  8D 8B C0  STA $C08B
       BF53-  4C D8 FF  JMP $FFD8
```

Of course, before coming back via the RTI, ProDOS tries to USE the interrupt. If you have set up one or more interrupt vectors with the ProDOS system call, they will be called.

INT.SPLICE restores the contents of $45 and switches the main $D000 bank on. Then it jumps back to $BFD3 with the information about which $D000 bank really should be on. $BFD3 turns on the other bank if necessary, and returns to the point at which the interrupt occured.

The instruction at $FFC8 is interesting. STA $C082 turns on the monitor ROM, so the next instruction to be executed is at $FFCB in ROM. This is an RTS opcode, so the address on the stack at that point is used. There are two possible values: $FA41 if an IRQ interrupt is being processed, or $FA61 if a RESET is being processed. This means the RTS will effectively branch to $FA42 or $FA62.

Uh Oh! At this point you had better hope that you are not running with the original Apple monitor ROM. The Apple II Plus ROM (called Autostart Monitor) and the Apple //e ROM are fine. $FA42 is the second instruction of the IRQ code, and $FA62 is the standard RESET handler. But the original ROM, like I have in my serial 219 machine, has entirely different code there.

I have an $FF at $FA42, followed by code for the monitor S (single step) command. And $FA62 is right in the middle of the S command. There is no telling what might happen, short of actually trying it out. No thanks. Just remember that RESET, BRK, and IRQ interrupts will not work correctly if they happen when the RAM area is switched on and you have the old original monitor in ROM.

There is another small empty space from $FFE9 through $FFF9, 17 bytes.

Perhaps I should point out that the listings this month and last are from the latest release of ProDOS, which may not be the final released version. However, I would expect any differences in the regions I have covered so far to be slight.

```
                  1000 *--------------------------------
                  1010 *SAVE S.PRODOS F800-FFFF
                  1020 *--------------------------------
003A-             1030 RUNNING.SUM        .EQ $3A
003A-             1040 TBUF.0             .EQ $3A
003B-             1050 BYTE.AT.BUF00      .EQ $3B
003C-             1060 BYTE.AT.BUF01      .EQ $3C
003D-             1070 LAST.BYTE          .EQ $3D
003E-             1080 SLOT.X16           .EQ $3E
003F-             1090 INDEX.OF.LAST.BYTE .EQ $3F
```

```
                     1100 *-------------------------------
0042-                1110 RWB.COMMAND .EQ $42
0043-                1120 RWB.SLOT    .EQ $43   DSSSXXXX
0044-                1130 RWB.BUFFER  .EQ $44,45
0046-                1140 RWB.BLOCK   .EQ $46,47   0...279
                     1150 *-------------------------------
4700-                1160 BUFF.BASE .EQ $4700 DUMMY ADDRESS FOR ASSEMBLY ONLY
                     1170 *-------------------------------
BF56-                1180 SAVE.LOC45 .EQ $BF56
BF57-                1190 SAVE.D000  .EQ $BF57
BF88-                1200 INTAREG    .EQ $BF88
BF8D-                1210 INTBANKID  .EQ $BF8D
BFD3-                1220 IRQXIT.3   .EQ $BFD3
                     1230 *-------------------------------
C080-                1240 DRV.PHASE   .EQ $C080
C088-                1250 DRV.MTROFF  .EQ $C088
C089-                1260 DRV.MTRON   .EQ $C089
C08A-                1270 DRV.ENBL.0  .EQ $C08A
C08C-                1280 DRV.Q6L     .EQ $C08C
C08D-                1290 DRV.Q6H     .EQ $C08D
C08E-                1300 DRV.Q7L     .EQ $C08E
C08F-                1310 DRV.Q7H     .EQ $C08F
                     1320 *-------------------------------
                     1330 *                   <<<COMPUTED >>>
0060-                1340 MODIFIER .EQ $60   <<<SLOT * 16>>>
                     1350 *-------------------------------
                     1360        .OR $F800
                     1370        .TA $800

                     2940 *-------------------------------
                     2950 SEEK.TRACK
F90C- 0A             2960        ASL           GET PHYSICAL TRACK #
F90D- 8D 6F FB       2970        STA HDR.TRACK      ...SAVE HERE
F910- 20 25 F9       2980        JSR CLEAR.PHASES  (CARRY WAS CLEAR)
F913- 20 F1 FC       2990        JSR GET.SSSD.IN.X
F916- BD 58 FB       3000        LDA OLD.TRACK.TABLE,X
F919- 8D 5A FB       3010        STA CURRENT.TRACK
F91C- AD 6F FB       3020        LDA HDR.TRACK
F91F- 9D 58 FB       3030        STA OLD.TRACK.TABLE,X
F922- 20 33 F9       3040        JSR SEEK.TRACK.ABSOLUTE
                     3050 *-------------------------------
                     3060 CLEAR.PHASES
F925- A0 03          3070        LDY #3
F927- 98             3080 .1     TYA
F928- 20 8A F9       3090        JSR PHASE.COMMANDER
F92B- 88             3100        DEY
F92C- 10 F9          3110        BPL .1
F92E- 4E 5A FB       3120        LSR CURRENT.TRACK    BACK TO LOGICAL TRACK #
F931- 18             3130        CLC           SIGNAL NO ERROR
F932- 60             3140        RTS
                     3150 *-------------------------------
                     3160 SEEK.TRACK.ABSOLUTE
F933- 8D 72 FB       3170        STA TARGET.TRACK  SAVE ACTUAL TRACK #
F936- CD 5A FB       3180        CMP CURRENT.TRACK ALREADY THERE?
F939- F0 4C          3190        BEQ .7            ...YES
F93B- A9 00          3200        LDA #0
F93D- 8D 6B FB       3210        STA STEP.CNT      # STEPS SO FAR
F940- AD 5A FB       3220 .1     LDA CURRENT.TRACK
F943- 8D 71 FB       3230        STA CURRENT.TRACK.OLD
F946- 38             3240        SEC
F947- ED 72 FB       3250        SBC TARGET.TRACK
F94A- F0 37          3260        BEQ .6            ...WE HAVE ARRIVED
F94C- B0 07          3270        BCS .2            CURRENT > DESIRED
F94E- 49 FF          3280        EOR #$FF          CURRENT < DESIRED
F950- EE 5A FB       3290        INC CURRENT.TRACK
F953- 90 05          3300        BCC .3            ...ALWAYS
F955- 69 FE          3310 .2     ADC #$FE          .CS., SO A=A-1
F957- CE 5A FB       3320        DEC CURRENT.TRACK
F95A- CD 6B FB       3330 .3     CMP STEP.CNT GET MINIMUM OF:
F95D- 90 03          3340        BCC .4         1. # OF TRACKS TO MOVE LESS 1
F95F- AD 6B FB       3350        LDA STEP.CNT   2. # OF STEPS SO FAR
F962- C9 09          3360 .4     CMP #9         3. EIGHT
F964- B0 02          3370        BCS .5
F966- A8             3380        TAY
F967- 38             3390        SEC           TURN NEW PHASE ON
F968- 20 87 F9       3400 .5     JSR .7
F96B- B9 73 FB       3410        LDA ONTBL,Y  DELAY
F96E- 20 85 FB       3420        JSR DELAY.100
F971- AD 71 FB       3430        LDA CURRENT.TRACK.OLD
F974- 18             3440        CLC           TURN OLD PHASE OFF
```

```
F975- 20 8A F9             JSR PHASE.COMMANDER
F978- B9 7C FB             LDA OFFTBL,Y DELAY
F97B- 20 85 FB             JSR DELAY.100
F97E- EE 6B FB             INC STEP.CNT # OF STEPS SO FAR
F981- D0 BD                BNE .1         ...ALWAYS
F983- 20 85 FB     .6      JSR DELAY.100
F986- 18                   CLC            TURN PHASE OFF
F987- AD 5A FB     .7      LDA CURRENT.TRACK
                   *-------------------------------
                   *       (A) = TRACK #
                   *       .CC. THEN PHASE OFF
                   *       .CS. THEN PHASE ON
                   *-------------------------------
                   PHASE.COMMANDER
F98A- 29 03                AND #3         ONLY KEEP LOWER TWO BITS
F98C- 2A                   ROL                 00000XXC
F98D- 05 3E                ORA SLOT.X16        0SSS0XXC
F98F- AA                   TAX
F990- BD 80 C0             LDA DRV.PHASE,X
F993- A6 3E                LDX SLOT.X16        RESTORE SLOT*16
F995- 60                   RTS

                   *-------------------------------
                   WRITE.SECTOR
FD00- 38                   SEC            IN CASE WRITE-PROTECTED
FD01- BD 8D C0             LDA DRV.Q6H,X
FD04- BD 8E C0             LDA DRV.Q7L,X
FD07- 10 03                BPL .1         ...NOT WRITE PROTECTED
FD09- 4C DF FD             JMP WS.RET     ...PROTECTED, ERROR
                   *-------------------------------
FD0C- AD 00 FB     .1      LDA TBUF
FD0F- 85 3A                STA TBUF.0
                   *---WRITE 5 SYNC BYTES-----------
FD11- A9 FF                LDA #$FF
FD13- 9D 8F C0             STA DRV.Q7H,X
FD16- 1D 8C C0             ORA DRV.Q6L,X
FD19- A0 04                LDY #4
FD1B- EA                   NOP            $FF AT 40-CYCLE INTERVALS LEAVES
FD1C- 48                   PHA            TWO ZERO-BITS AFTER EACH $FF
FD1D- 68                   PLA
FD1E- 48           .2      PHA
FD1F- 68                   PLA
FD20- 20 E7 FD             JSR WRITE2
FD23- 88                   DEY
FD24- D0 F8                BNE .2
                   *---WRITE $D5 AA AD HEADER-------
FD26- A9 D5                LDA #$D5
FD28- 20 E6 FD             JSR WRITE1
FD2B- A9 AA                LDA #$AA
FD2D- 20 E6 FD             JSR WRITE1
FD30- A9 AD                LDA #$AD
FD32- 20 E6 FD             JSR WRITE1
                   *---WRITE 86 BYTES FROM TBUF------------------
                   *---BACKWARDS:   TBUF+85...TBUF+1, TBUF.0------
FD35- 98                   TYA            =0
FD36- A0 56                LDY #86
FD38- D0 03                BNE .4
FD3A- B9 00 FB     .3      LDA TBUF,Y
FD3D- 59 FF FA     .4      EOR TBUF-1,Y
FD40- AA                   TAX
FD41- BD 03 FA             LDA BIT.PAIR.TABLE+3,X
FD44- A6 3E                LDX SLOT.X16
FD46- 9D 8D C0             STA DRV.Q6H,X
FD49- BD 8C C0             LDA DRV.Q6L,X
FD4C- 88                   DEY
FD4D- D0 EB                BNE .3
FD4F- A5 3A                LDA TBUF.0
                   *---WRITE PORTION OF BUFFER------
                   *---UP TO A PAGE BOUNDARY--------
FD51- A0 00                LDY #*-*       FILLED IN WITH LO-BYTE OF BUFFER ADDRESS
FD53- 59 00 47     WS...5  EOR BUFF.BASE,Y   HI-BYTE FILLED IN
FD56- 29 FC                AND #$FC
FD58- AA                   TAX
FD59- BD 03 FA             LDA BIT.PAIR.TABLE+3,X
FD5C- A2 60        WS...6  LDX #MODIFIER
FD5E- 9D 8D C0             STA DRV.Q6H,X
FD61- BD 8C C0             LDA DRV.Q6L,X
FD64- B9 00 47     WS...7  LDA BUFF.BASE,Y     HI-BYTE FILLED IN
FD67- C8                   INY
FD68- D0 E9                BNE WS...5
```

```
                   *---BRANCH ACCORDING TO BUFFER BOUNDARY CONDITIONS-----
FD6A- A5 3B                     LDA BYTE.AT.BUF00
FD6C- F0 52                     BEQ WS..17    ...BUFFER ALL IN ONE PAGE
FD6E- A5 3F                     LDA INDEX.OF.LAST.BYTE
FD70- F0 41                     BEQ WS..16    ...ONLY ONE BYTE IN NEXT PAGE
                   *---MORE THAN ONE BYTE IN NEXT PAGE-------------------
FD72- 4A                        LSR           ...DELAY TWO CYCLES
FD73- A5 3B                     LDA BYTE.AT.BUF00  PRE.NYBBLE ALREADY ENCODED
FD75- 9D 8D C0                  STA DRV.Q6H,X      THIS BYTE
FD78- BD 8C C0                  LDA DRV.Q6L,X
FD7B- A5 3C                     LDA BYTE.AT.BUF01
FD7D- EA                        NOP
FD7E- C8                        INY
FD7F- B0 18                     BCS WS..12
FD81- 59 00 48          WS...8  EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
FD84- 29 FC                     AND #$FC
FD86- AA                        TAX
FD87- BD 03 FA                  LDA BIT.PAIR.TABLE+3,X
FD8A- A2 60             WS...9  LDX #MODIFIER
FD8C- 9D 8D C0                  STA DRV.Q6H,X
FD8F- BD 8C C0                  LDA DRV.Q6L,X
FD92- B9 00 48          WS..10  LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
FD95- C8                        INY
FD96- 59 00 48          WS..11  EOR BUFF.BASE+256,Y    HI-BYTE FILLED IN
FD99- C4 3F             WS..12  CPY INDEX.OF.LAST.BYTE
FD9B- 29 FC                     AND #$FC
FD9D- AA                        TAX
FD9E- BD 03 FA                  LDA BIT.PAIR.TABLE+3,X
FDA1- A2 60             WS..13  LDX #MODIFIER
FDA3- 9D 8D C0                  STA DRV.Q6H,X
FDA6- BD 8C C0                  LDA DRV.Q6L,X
FDA9- B9 00 48          WS..14  LDA BUFF.BASE+256,Y    HI-BYTE FILLED IN
FDAC- C8                        INY
FDAD- 90 D2                     BCC WS...8
FDAF- B0 00                     BCS .15       ...3 CYCLE NOP
FDB1- B0 0D             .15     BCS WS..17    ...ALWAYS
                   *---WRITE BYTE AT BUFFER.00--------------------------
FDB3- AD 3B 00          WS..16  .DA #$AD,BYTE.AT.BUF00    4 CYCLES: LDA BYTE.AT.BUF00
FDB6- 9D 8D C0                  STA DRV.Q6H,X
FDB9- BD 8C C0                  LDA DRV.Q6L,X
FDBC- 48                        PHA
FDBD- 68                        PLA
FDBE- 48                        PHA
FDBF- 68                        PLA
FDC0- A6 3D             WS..17  LDX LAST.BYTE
FDC2- BD 03 FA                  LDA BIT.PAIR.TABLE+3,X
FDC5- A2 60             WS..18  LDX #MODIFIER
FDC7- 9D 8D C0                  STA DRV.Q6H,X
FDCA- BD 8C C0                  LDA DRV.Q6L,X
FDCD- A0 00                     LDY #0
FDCF- 48                        PHA
FDD0- 68                        PLA
                   *---WRITE DATA TRAILER:  $DE AA EB FF----------
FDD1- EA                        NOP
FDD2- EA                        NOP
FDD3- B9 C4 F9          .19     LDA DATA.TRAILER,Y
FDD6- 20 E9 FD                  JSR WRITE3
FDD9- C8                        INY
FDDA- C0 04                     CPY #4
FDDC- D0 F5                     BNE .19
FDDE- 18                        CLC           SIGNAL NO ERROR
FDDF- BD 8E C0          WS.RET  LDA DRV.Q7L,X     DRIVE TO SAFE MODE
FDE2- BD 8C C0                  LDA DRV.Q6L,X
FDE5- 60                        RTS
                   *--------------------------------
FDE6- 18                WRITE1  CLC
FDE7- 48                WRITE2  PHA
FDE8- 68                        PLA
FDE9- 9D 8D C0          WRITE3  STA DRV.Q6H,X
FDEC- 1D 8C C0                  ORA DRV.Q6L,X
FDEF- 60                        RTS
                   *--------------------------------
                        PRE.NYBBLE
FDF0- A5 44                     LDA RWB.BUFFER    PLUG IN ADDRESS TO LOOP BELOW
FDF2- A4 45                     LDY RWB.BUFFER+1
FDF4- 18                        CLC
FDF5- 69 02                     ADC #2
FDF7- 90 01                     BCC .1
FDF9- C8                        INY
```

```
FDFA- 8D 30 FE 8490 .1      STA PN...6+1
FDFD- 8C 31 FE 8500         STY PN...6+2
FE00- 38       8510         SEC
FE01- E9 56    8520         SBC #$56
FE03- B0 01    8530         BCS .2
FE05- 88       8540         DEY
FE06- 8D 25 FE 8550 .2      STA PN...5+1
FE09- 8C 26 FE 8560         STY PN...5+2
FE0C- 38       8570         SEC
FE0D- E9 56    8580         SBC #$56
FE0F- B0 01    8590         BCS .3
FE11- 88       8600         DEY
FE12- 8D 1B FE 8610 .3      STA PN...4+1
FE15- 8C 1C FE 8620         STY PN...4+2
               8630 *---PACK THE LOWER TWO BITS INTO TBUF-------------
FE18- A0 AA    8640         LDY #170
FE1A- B9 56 46 8650 PN...4 LDA BUFF.BASE-170,Y   ADDRESS FILLED IN
FE1D- 29 03    8660         AND #3
FE1F- AA       8670         TAX
FE20- BD E0 F9 8680         LDA BIT.PAIR.RIGHT,X
FE23- 48       8690         PHA
FE24- B9 AC 46 8700 PN...5 LDA BUFF.BASE-84,Y
FE27- 29 03    8710         AND #3
FE29- AA       8720         TAX
FE2A- 68       8730         PLA
FE2B- 1D C0 F9 8740         ORA BIT.PAIR.MIDDLE,X
FE2E- 48       8750         PHA
FE2F- B9 02 47 8760 PN...6 LDA BUFF.BASE+2,Y
FE32- 29 03    8770         AND #3
FE34- AA       8780         TAX
FE35- 68       8790         PLA
FE36- 1D A0 F9 8800         ORA BIT.PAIR.LEFT,X
FE39- 48       8810         PHA
FE3A- 98       8820         TYA
FE3B- 49 FF    8830         EOR #$FF
FE3D- AA       8840         TAX
FE3E- 68       8850         PLA
FE3F- 9D 00 FB 8860         STA TBUF,X
FE42- C8       8870         INY
FE43- D0 D5    8880         BNE PN...4
               8890 *---DETERMINE BUFFER BOUNDARY CONDITIONS----------
               8900 *---AND SETUP WRITE.SECTOR ACCORDINGLY------------
FE45- A4 44    8910         LDY RWB.BUFFER
FE47- 88       8920         DEY
FE48- 84 3F    8930         STY INDEX.OF.LAST.BYTE
FE4A- A5 44    8940         LDA RWB.BUFFER
FE4C- 8D 52 FD 8950         STA WS...5-1
FE4F- F0 0E    8960         BEQ .7
FE51- 49 FF    8970         EOR #$FF
FE53- A8       8980         TAY
FE54- B1 44    8990         LDA (RWB.BUFFER),Y
FE56- C8       9000         INY
FE57- 51 44    9010         EOR (RWB.BUFFER),Y
FE59- 29 FC    9020         AND #$FC
FE5B- AA       9030         TAX
FE5C- BD 03 FA 9040         LDA BIT.PAIR.TABLE+3,X
FE5F- 85 3B    9050 .7      STA BYTE.AT.BUF00    =0 IF BUFFER NOT SPLIT
FE61- F0 0C    9060         BEQ .9
FE63- A5 3F    9070         LDA INDEX.OF.LAST.BYTE
FE65- 4A       9080         LSR
FE66- B1 44    9090         LDA (RWB.BUFFER),Y
FE68- 90 03    9100         BCC .8
FE6A- C8       9110         INY
FE6B- 51 44    9120         EOR (RWB.BUFFER),Y
FE6D- 85 3C    9130 .8      STA BYTE.AT.BUF01
FE6F- A0 FF    9140 .9      LDY #$FF
FE71- B1 44    9150         LDA (RWB.BUFFER),Y
FE73- 29 FC    9160         AND #$FC
FE75- 85 3D    9170         STA LAST.BYTE
               9180 *---INSTALL BUFFER ADDRESSES IN WRITE.SECTOR------
FE77- A4 45    9190         LDY RWB.BUFFER+1
FE79- 8C 55 FD 9200         STY WS...5+2
FE7C- 8C 66 FD 9210         STY WS...7+2
FE7F- C8       9220         INY
FE80- 8C 83 FD 9230         STY WS...8+2
FE83- 8C 94 FD 9240         STY WS..10+2
FE86- 8C 98 FD 9250         STY WS..11+2
FE89- 8C AB FD 9260         STY WS..14+2
```

```
                    9270 *---INSTALL SLOT*16 IN WRITE.SECTOR---------------
FE8C- A6 3E         9280        LDX SLOT.X16
FE8E- 8E 5D FD      9290        STX WS...6+1
FE91- 8E 8B FD      9300        STX WS...9+1
FE94- 8E A2 FD      9310        STX WS..13+1
FE97- 8E C6 FD      9320        STX WS..18+1
FE9A- 60            9330        RTS
                    9340 *--------------------------------
                    9350 WAIT.FOR.OLD.MOTOR.TO.STOP
FE9B- 4D 59 FB      9360        EOR OLD.SLOT       SAME SLOT AS BEFORE?
FE9E- 0A            9370        ASL                (IGNORE DRIVE)
FE9F- F0 1C         9380        BEQ .2             ...YES
FEA1- A9 01         9390        LDA #1        LONG MOTOR.TIME
FEA3- 8D 70 FB      9400        STA MOTOR.TIME+1  (COUNTS BACKWARDS)
FEA6- AD 59 FB      9410 .1     LDA OLD.SLOT
FEA9- 29 70         9420        AND #$70
FEAB- AA            9430        TAX
FEAC- F0 0F         9440        BEQ .2        ...NO PREVIOUS MOTOR RUNNING
FEAE- 20 DC FC      9450        JSR CHECK.IF.MOTOR.RUNNING.X
FEB1- F0 0A         9460        BEQ .2        ...NOT RUNNING YET
FEB3- A9 01         9470        LDA #1        DELAY ANOTHER 100 USECS
FEB5- 20 85 FB      9480        JSR DELAY.100
FEB8- AD 70 FB      9490        LDA MOTOR.TIME+1
FEBB- D0 E9         9500        BNE .1        KEEP WAITING
FEBD- 60            9510 .2     RTS
                    9520 *--------------------------------
FEBE-               9530        .BS $FF9B-*        <<<<EMPTY SPACE>>>>
                    9540 *--------------------------------
                    9550 IRQ
FF9B- 48            9560        PHA           SAVE A-REG
FF9C- A5 45         9570        LDA $45      SAVE LOC $45
FF9E- 8D 56 BF      9580        STA SAVE.LOC45
FFA1- 68            9590        PLA           SAVE A-REG AT LOC $45
FFA2- 85 45         9600        STA $45
FFA4- 68            9610        PLA           GET STATUS BEFORE IRQ
FFA5- 48            9620        PHA
FFA6- 29 10         9630        AND #$10      SEE IF "BRK"
FFA8- D0 18         9640        BNE .2        ...YES, LET MONITOR DO IT
FFAA- AD 00 D0      9650        LDA $D000     SAVE $D000 BANK ID
FFAD- 49 D8         9660        EOR #$D8
FFAF- F0 02         9670        BEQ .1
FFB1- A9 FF         9680        LDA #$FF
FFB3- 8D 8D BF      9690 .1     STA INTBANKID
FFB6- 8D 57 BF      9700        STA SAVE.D000
FFB9- A9 BF         9710        LDA #$BF      PUSH FAKE "RTI" VECTOR WITH
FFBB- 48            9720        PHA                IRQ DISABLED
FFBC- A9 50         9730        LDA #$50           AND SET TO RETURN TO $BF50
FFBE- 48            9740        PHA
FFBF- A9 04         9750        LDA #4
FFC1- 48            9760        PHA
FFC2- A9 FA         9770 .2     LDA #$FA      PUSH "RTS" VECTOR FOR MONITOR
FFC4- 48            9780        PHA
FFC5- A9 41         9790        LDA #$41
FFC7- 48            9800        PHA
                    9810 CALL.MONITOR
FFC8- 8D 82 C0      9820        STA $C082     SWITCH TO MOTHERBOARD
                    9830 *--------------------------------
                    9840 RESET
FFCB- AD D7 FF      9850        LDA RESET.VECTOR+1
FFCE- 48            9860        PHA           PUSH "RTS" VECTOR FOR MONITOR
FFCF- AD D6 FF      9870        LDA RESET.VECTOR
FFD2- 48            9880        PHA
FFD3- 4C C8 FF      9890        JMP CALL.MONITOR
                    9900 *--------------------------------
                    9910 RESET.VECTOR
FFD6- 61 FA         9920        .DA $FA61     MON.RESET-1
                    9930 *--------------------------------
                    9940 INT.SPLICE
FFD8- 8D 88 BF      9950        STA INTAREG
FFDB- AD 56 BF      9960        LDA SAVE.LOC45
FFDE- 85 45         9970        STA $45
FFE0- AD 8B C0      9980        LDA $C08B     SWITCH TO MAIN $D000 BANK
FFE3- AD 57 BF      9990        LDA SAVE.D000
FFE6- 4C D3 BF      10000        JMP IRQXIT.3
                    10010 *--------------------------------
FFE9-               10020        .BS $FFFA-*        <<<<<EMPTY SPACE>>>>>
                    10030 *--------------------------------
FFFA- FB 03         10040 V.NMI      .DA $03FB
FFFC- CB FF         10050 V.RESET    .DA RESET
FFFE- 9B FF         10060 V.IRQ      .DA IRQ
```

More Assembly Listing into Text Files...........Tracy L. Shafer
MacDill AFB, FL

In the October '83 issue of AAL, Robert F. O'Brien presented a way to create a text file containing the assembly listing of a large program. (See also "Assembly Listing Into a Text File", by Bill Morgan, July '83 AAL.) Actually, he created several text files; one for each .IN directive in the root file. You can't put the whole listing into one text file by using one .TF directive because of the way the .IN directive affects the DOS I/O hooks.

Robert's method for obtaining assembly listing text files is good, but I found a different way to create the text files of assembly listings that doesn't involve creating separate SYMBOLS sections, deleting duplicate labels, and putting up with "EXTRA DEFINITIONS ERROR" messages. It's a fairly simple approach and hinges on the fact that the problem presented by the .IN directive affects the source file containing the .IN, but not the source file to which the .IN refers. Instead of putting one .TF directive in the root file, put a .TF in each source file pointed to by a .IN directive.

For example:

```
ROOT FILE

1000    .DU
1010    .IN PART 1
1020    .IN PART 2
1030    .ED

PART 1

1000    .TF LISTING 1
1010    (source for part 1)

PART 2

1000    .TF LISTING 2
1010    (source for part 2)
```

From here on, follow Bill Morgan's original instructions. What follows is a summary of those instructions.

After deleting all other .TF directives, or turning them into comments by inserting "*" at the beginning of the line, typing ASM will create two binary files named LISTING 1 and LISTING 2. Each of these contains the assembly listing of PART 1 and PART 2 respectively, in text form. These binary files will not have starting address and length in the first four bytes. DO NOT attempt to BLOAD these files. You could really clobber DOS. To obtain true text files, make the following patch to the S-C Assembler before you assemble the program:

```
$1000 versions:  $29DF:0  (original value is 04)
$D000 versions:  $C083 C083 EAF9:0 N C083
```

After the patch is made, assemble the program and restore the original value to $29DF ($EAF9).

For really large programs, it could get very tedious adding a .TF directive to each sub-file to obtain a text file listing and then deleting those .TF directives to prevent messing up the object file the next time the program is assembled. Fortunately, the S-C Macro Assembler's conditional assembly feature makes our work a lot easier. By placing an equated flag in the root file and surrounding each .TF with .DO and .FIN, we only have to change one line to set up our program for text file output or object file creation. For example:

```
ROOT FILE

1000 LSTOUT .EQ 0       TO ASSEMBLE OBJECT
1010 *          1       TO OUTPUT TEXT FILES
1020        .DO LSTOUT
1040        .DU
1050        .ELSE
1060        .TF OBJECT FILE
1070        .FIN
1080        .IN PART 1
1090        .IN PART 2
1100        .DO LSTOUT
1110        .ED
1120        .FIN

PART 1

1000        .DO LSTOUT
1010        .TF LISTING 1
1020        .FIN
1030        (source for part 1)

PART 2

1000        .DO LSTOUT
1010        .TF LISTING 2
1020        .FIN
1030        (source for part 2)
```

Don't forget to patch $29DF ($EAF9 for the language card version) with 0 to output true text files and back to 4 create object files. The last thing to remember is to use .LIST ON during the assembly. You won't write any text files if the assembler isn't producing a listing.

Note on Aztec C....................................Bill Morgan

I just talked to the people at Manx Software about the ProDOS version of their C compiler, and this time they assured me that owners of the current Apple DOS version will be able to purchase the ProDOS version at a reduced upgrade price. That is enough to tip the balance in favor of buying the compiler right now, so I have ordered some. List price is $199: we'll have them for $180 + shipping.

Generalized GOTO and GOSUB................Bob Sander-Cederlof

Tim Mowchanuk, a lecturer at Brisbane College in Australia, sent the following suggestion:

> "How can I implement a named GOTO or GOSUB routine? There are numerous routines that implement computed GOTO/GOSUB, but I consider that a futile exercise. Computed GOTO/GOSUB mess up renumbering utilities, and violate modern trends toward structured programming.
>
> "What I really want is something that will handle BASIC like
>
> ```
> 100 & GOSUB NAME$
> ```
>
> where NAME$ holds the name of a subroutine. I envision subroutine names being defined by a special REM statement of the form
>
> ```
> 200 REM "SUBROUTINE NAME"
> ```
>
> The &GOSUB or &GOTO processor can search through the program for a line beginning with a REM token. If the first non-blank after the REM token is a quotation mark, the processor can compare the characters to the string value. If there is an exact match, the line containing the REM is the target for the &GOTO or &GOSUB."

The problem sounded just the right size for an interesting AAL article, so I started trying to write some code.

I published an &GOSUB routine back in April 1981 of the type that Tim thinks futile. The following program combines the two "futile" computed &GOSUB and &GOTO routines with two new ones that allow the computed value to be a string expression. If the expression after &GOTO or &GOSUB is numeric, the processor will search for a matching line number. If the result is a string, the processor will search for a REM label as Tim described above.

Only REM's at the beginning of a numbered line will be considered as labels. The label must be included in quotation marks. Spaces are OK between the word REM and the first quotation mark. Anything after the second quotation mark will be ignored.

You can now write a menu program that uses the actual command word as the name of a subroutine, and cease worrying about line numbers. The accompanying Applesoft program is an example of just such a technique.

```
100  PRINT  CHR$ (4)"BLOAD B.LABELLED GO'S": CALL 768
1000  DATA  SEND,RECEIVE,EDIT,LOAD,SAVE,EXIT,.
1010 I = 0
1020 I = I + 1: READ A$(I): IF A$(I) <  > "." THEN 1020
1030 N = I - 1
1100  INPUT C$:I = 0
1110 I = I + 1: IF C$ = A$(I) THEN  &  GOSUB C$: GOTO 1100
1120  IF I < N THEN 1110
1130  PRINT "NO SUCH COMMAND": GOTO 1100
```

```
2000  REM "SEND"
2010  PRINT "SEND NOT YET IMPLEMENTED": RETURN
2500  REM "RECEIVE"
2510  PRINT "RECEIVE IS NOT READY": RETURN
3000  REM "EDIT"
3010  PRINT "MAYBE YOU CAN EDIT LATER": RETURN
3500  REM "LOAD"
3510  PRINT "LOAD WHAT, WHERE, HOW?": RETURN
4000  REM "SAVE"
4010  PRINT "SAVE WHAT, WHERE, HOW": RETURN
4500  REM "EXIT"
4510  PRINT "AH!  THAT I CAN DO!": POP : END
```

```
                  1000 *SAVE S.LABELLED GO'S
                  1010 *--------------------------------
                  1020 *       & GOTO <STR EXP>
                  1030 *       & GOSUB<STR EXP>
                  1040 *       REM "<LABEL>"
                  1050 *
                  1060 *       AS SUGGESTED BY TIM MOWCHANUK
                  1070 *--------------------------------
0011-             1080 AS.VALTYP  .EQ $11
0052-             1090 AS.TEMPPT  .EQ $52,53
005E-             1100 INDEX.REM  .EQ $5E
005F-             1110 INDEX.GO   .EQ $5F
0067-             1120 PRGBOT     .EQ $67,68
0075-             1130 AS.CURLIN  .EQ $75,76
009B-             1140 PNTR       .EQ $9B,9C
009D-             1150 STRLEN     .EQ $9D
009E-             1160 STRADR     .EQ $9E,9F
00A0-             1170 VPNT       .EQ $A0,A1
00B8-             1180 TXTPTR     .EQ $B8,B9
                  1190 *--------------------------------
00AB-             1200 TKN.GOTO   .EQ $AB
00B0-             1210 TKN.GOSUB  .EQ $B0
00B2-             1220 TKN.REM    .EQ $B2
                  1230 *--------------------------------
03F5-             1240 AMPERSAND.VECTOR .EQ $3F5 ... 3F7
                  1250 *--------------------------------
00B1-             1260 AS.CHRGET  .EQ $00B1
00B7-             1270 AS.CHRGOT  .EQ $00B7
D3D6-             1280 AS.MEMCHK  .EQ $D3D6
D7D2-             1290 AS.NEWSTT  .EQ $D7D2
D941-             1300 AS.GOTO1   .EQ $D941
D95E-             1310 AS.GOTO.3  .EQ $D95E
D97C-             1320 AS.UNDERR  .EQ $D97C
DD7B-             1330 AS.FRMEVL  .EQ $DD7B
DEC9-             1340 AS.SYNERR  .EQ $DEC9
E604-             1350 AS.FRETMP  .EQ $E604
E752-             1360 AS.GETADR  .EQ $E752
                  1370 *--------------------------------
                  1380        .OR $300
                  1390        .TF B.LABELLED GO'S
                  1400 *--------------------------------
0300- A9 0B       1410 SETUP  LDA #LABELLED.GOTO.AND.GOSUB
0302- 8D F6 03    1420        STA AMPERSAND.VECTOR+1
0305- A9 03       1430        LDA /LABELLED.GOTO.AND.GOSUB
0307- 8D F7 03    1440        STA AMPERSAND.VECTOR+2
030A- 60          1450        RTS
                  1460 *--------------------------------
                  1470 LABELLED.GOTO.AND.GOSUB
030B- 20 B7 00    1480        JSR AS.CHRGOT
030E- C9 AB       1490        CMP #TKN.GOTO
0310- F0 1D       1500        BEQ .3
0312- C9 B0       1510        CMP #TKN.GOSUB
0314- F0 03       1520        BEQ .2        ...GOOD SYNTAX SO FAR
0316- 4C C9 DE    1530 .1     JMP AS.SYNERR
                  1540 *---SETUP GOSUB RETURN DATA------
0319- A9 03       1550 .2     LDA #3
031B- 20 D6 D3    1560        JSR AS.MEMCHK
031E- A5 B9       1570        LDA TXTPTR+1
0320- 48          1580        PHA
0321- A5 B8       1590        LDA TXTPTR
0323- 48          1600        PHA
0324- A5 76       1610        LDA AS.CURLIN+1
0326- 48          1620        PHA
0327- A5 75       1630        LDA AS.CURLIN
0329- 48          1640        PHA
```

```
032A- A9 B0             LDA #TKN.GOSUB
032C- 48                PHA
032D- D0 02             BNE .4          ...ALWAYS
                  *---SETUP FOR GOTO--------------
032F- 68          .3    PLA             POP RETURN TO "NEWSTT"
0330- 68                PLA
                  *---FIND LABEL AFTER TOKEN-------
0331- 20 B1 00    .4    JSR AS.CHRGET
0334- F0 E0             BEQ .1
0336- 20 7B DD          JSR AS.FRMEVL      EVALUATE EXPRESSION
0339- 24 11             BIT AS.VALTYP      $00 IF NUMERIC, $FF IF STRING
033B- 30 10             BMI .5             ...STRING
                  *---NUMERIC EXPRESSION-----------
033D- 20 52 E7          JSR AS.GETADR      CONVERT TO INTEGER
0340- 20 41 D9          JSR AS.GOTO1
0343- 4C D2 D7          JMP AS.NEWSTT
                  *---FREE ANY TEMP STRINGS--------
0346- A5 53       .45   LDA AS.TEMPPT+1
0348- A0 00             LDY #0
034A- 20 04 E6          JSR AS.FRETMP
034D- A5 52       .5    LDA AS.TEMPPT
034F- C9 56             CMP #$56        EMPTY?
0351- B0 F3             BCS .45         ...NO, FREE A STRING
                  *---COPY STRING LENGTH/ADDRESS---
0353- A0 02             LDY #2
0355- B1 A0       .55   LDA (VPNT),Y
0357- 99 9D 00          STA STRLEN,Y
035A- 88                DEY
035B- 10 F8             BPL .55
                  *---SEARCH PROGRAM FOR LABEL-----
035D- A5 68             LDA PRGBOT+1       POINT TO BEGINNING
035F- A6 67             LDX PRGBOT            OF PROGRAM
                  *---LOOK AT NEXT LINE------------
0361- 85 9C       .6    STA PNTR+1      UPDATE PNTR TO NEXT LINE
0363- 86 9B             STX PNTR
0365- A0 01             LDY #1          HI-BYTE OF FWD PNTR
0367- B1 9B             LDA (PNTR),Y
0369- F0 43             BEQ .11         ...END OF PROGRAM
                  *---CHECK FOR 'REM "'------------
036B- A0 04             LDY #4
036D- B1 9B             LDA (PNTR),Y
036F- C9 B2             CMP #TKN.REM
0371- D0 31             BNE .10         ...NOT REM STATEMENT
0373- C8          .7    INY             NEXT BYTE OF LINE
0374- B1 9B             LDA (PNTR),Y
0376- C9 20             CMP #' '        IGNORE BLANKS BEFORE "
0378- F0 F9             BEQ .7
037A- C9 22             CMP #'"'        " YET?
037C- D0 26             BNE .10         ...NO, NOT A LABEL
                  *---COMPARE LABEL----------------
037E- 84 5E             STY INDEX.REM
0380- A9 FF             LDA #-1
0382- 85 5F             STA INDEX.GO
0384- E6 5E       .8    INC INDEX.REM
0386- A4 5E             LDY INDEX.REM
0388- B1 9B             LDA (PNTR),Y
038A- F0 8A             BEQ .1          ...EARLY END OF LABEL
038C- E6 5F             INC INDEX.GO
038E- A4 5F             LDY INDEX.GO
0390- C9 22             CMP #'"'        LEGAL END OF LABEL?
0392- F0 06             BEQ .9          ...YES
0394- D1 9E             CMP (STRADR),Y
0396- F0 EC             BEQ .8          ...KEEP MATCHING
0398- D0 0A             BNE .10         ...DOESN'T MATCH
039A- C4 9D       .9    CPY STRLEN      CORRECT LENGTH?
039C- D0 06             BNE .10         ...NO, KEEP SEARCHING
                  *---FOUND LABEL, SO GO TO IT-----
039E- 20 5E D9          JSR AS.GOTO.3
03A1- 4C D2 D7          JMP AS.NEWSTT
                  *---DOESN'T MATCH, TRY NEXT LINE-
03A4- A0 00       .10   LDY #0          GET FORWARD POINTER
03A6- B1 9B             LDA (PNTR),Y       LO-BYTE
03A8- AA                TAX
03A9- C8                INY                HI-BYTE
03AA- B1 9B             LDA (PNTR),Y
03AC- D0 B3             BNE .6          ...NOT END OF PROGRAM YET
                  *---END OF PROGRAM, UNDEF LBL----
03AE- 4C 7C D9    .11   JMP AS.UNDERR
```

Timemaster II from Applied Engineering.....Bob Sander-Cederlof

It may come as a surprise (it did to me), but there are apparently now only three calendar/clocks still on the market for the Apple II, II Plus, //e. The others, and there were a lot of them, seemed to have dropped off the map. And even one of the three (Mountain Computer) does not advertise anywhere I can find.

Another surprise: the most expensive clock has the fewest features, and the least expensive has the most features.

Mountain Computer Apple Clock

$280 in current catalog listing; most recent ad I could find was in Jan 1980 Byte, at $199. Features below are guessed at from ad and conversations with Dan Pote. Works with BASIC only, does not include any DOS Dater or ProDOS support.

Gives month, day of month, hour, minute, second, millisecond

Interrupt available: Second, Millisecond

Thunderware Thunderclock Plus

Gives month, day of month, day of week, hour, minute, second.

$150 with BASIC software for DOS or ProDOS
$ 29 extra for Pascal software
$ 29 extra for DOS-DATER/DEMO disk

Interrupts available: 64, 256, or 2048 times per second

Applied Engineering Timemaster

$129 includes Applesoft support for DOS or ProDOS
includes Pascal and CP/M support
includes DOS Dater

Gives year, month, day of month, day of week, hour, minute, second

Interrupts available: Millisecond, Second, Minute, Hour. Switchable to either NMI or IRQ interrupt line.

For some reason they have not chosen to explain, the wizards at Apple who created ProDOS decided to "wire in" support for the Thunderclock (and ONLY Thunderclock). A system call reads the time and date from Thunderclock, calculates the year from the given information, and stores year-month-day-hour-minute in a packed format at $BF90...BF93. ProDOS automatically records time/date of creation and time/date of last modification.

In order to get the year with these dates, ProDOS goes through a calculation to derive year from given day of month, month, and day of week information. The calculation involves remaindering and table lookup...but it only works from 1982 through 1987. I suppose by 1988 they will have generated a new version which works beyond, or else we won't care anymore. Better yet, by 1988 maybe they will have driver-ized the clock support so we can use Dan's card directly.

Dan Pote sent me a Timemaster to play with, in hopes that I would figure out how to make it look like a Thunderclock to ProDOS. I did, so if you buy one now it will be completely compatible with ProDOS. You select by DIP Switch which page of the onboard EPROM will be mapped into the $CN00 space (where N is slot 1-7). One setting selects the ProDOS section, and the others select various versions designed for use with DOS and Applesoft.

You can talk to Dan's card directly, as well as through the EPROM. If you don't like the way his firmware works (unlikely), you can either ignore it or change it.

(By the way.... Call A.P.P.L.E., a club/magazine with a penchant for value and quality, has chosen to offer another one of Applied Engineering's boards in its latest catalog: the Viewmaster 80. Their price is $140, which is 20% below normal retail.)

Finding Trouble in a Big RAM Card..........Bob Sander-Cederlof

Last night (Monday, Nov 28th) I took home an Apple to do some spreadsheet work. I took home the most portable one, but first swapped RAM cards. I took the STB-128 out of my oldest Apple and put it into the Apple II Plus with the fewest attachments.

When I plugged it in at home and booted the spreadsheet program, all appeared to be well. But it wasn't. I loaded in a model, and during the re-calculation the spreadsheet program hit a BRK opcode and died. I pressed RESET and looked at the partially re-calculated sheet: it was sprinkled with nonsense characters, and the keyboard was locked up. I played with various combinations for an hour or so, including other programs which use the RAM card. Everything pointed to there being a bad bit somewhere in the card.

Of course the RAM card test program was back at the office. I decided to write another one rather than face the two mile round trip.

The 128K space on the STB-128 is divided into 8 banks. You select a bank by storing a bank number (0-7) at any address in the $C080+slot*16 space which has bit 2 = 1. For slot 0, that means store in $C080, $C081, $C082, $C083, $C088, $C089, $C08A, or $C08B. The card has three green LEDs on top which show which bank is currently selected.

Each 16K bank is further divided to fit into the 12K address space between $D000 and $FFFF. The softswitch controlled by bit 3 in the $C08x address selects which of two 4K banks will be enabled at $D000-DFFF. The other 8K always sits at $E000-FFFF. A red LED signals which $D000 bank is selected.

The low-order two bits of the $C08x address control the mode of the RAM card. Accessing $C080 or $C088 write protects the card, and read enables it. This means the $D000-FFFF references the RAM card rather than the motherboard ROM. Accessing $C082 or $C08A write protects the RAM card and disables reading it; in other words, it switches on the motherboard ROM.

$C081 or $C089 also turn on the mother board ROM for reading, but if you access one of these twice in a row it will write enable the RAM card. In this mode reads reference the motherboard ROM, but writes write into the RAM card. This mode is used when loading the RAM card so that monitor and Applesoft routines which are in motherboard ROM can be used for the loading process.

Accessing $C083 or $C08B once read enables the STB-128 card and write protects it. A second access write enables the card. This is the mode we use for a memory test.

Thinking about how to test such a card, I wrote down the following "flow chart":

```
For Bank = 0 to 7
    Store Bank in $C083
    Access $C083 again to write enable
        Test $D000-DFFF
    Access $C08B twice
        Test $D000-FFFF
Next Bank
```

I broke the actual testing of a range of memory into four parts. First I stored zeroes into every location, and checked to be sure I read zeroes back. Then I did the same with $FF. Then, $55. Then, $AA. This is certainly not an exhaustive test, but I hoped it would be sufficient.

The tricky part was informing myself of the locations and values involved of any memory errors found during the test. I could not conveniently use the monitor subroutines to write addresses and values on the screen, because the monitor only existed in the motherboard ROM and it was switched off! So, I wrote a quick and dirty display routine.

The routine for display in the listing below is not quite so "quick and dirty". The program starts by clearing the screen using the monitor HOME subroutine at $FC58. Then it switches to the RAM card and runs the test. The program pokes test failure data directly to the screen. I direct the data for each of the 8 banks to a different line. When a failure occurs, I print the address, the value that should have been there, the actual value found, and the exclusive-or of the two values. The exclusive-or shows me which bit or bits was incorrect.

After running the test it was obvious that the least significant bit in banks 5 and 6 was not working. When it should be zero it was sometimes one, and vice versa.

I did not know which chip on the STB-128 card belonged to which bit slice or which bank, so I guessed. I was lucky, and guessed right the first time. I pulled out the chip I thought might be the bad one, and re-ran the test. This time the test indicated the least significant bit of banks 4-7 was missing. (It happened to be the chip in the lower-left corner when looking at the face of the card.)

I put the chip back in, hoping that it would miraculously heal itself. Then I looked at the back of the board to see if anything looked suspicious there. Sure enough! STB did not trim off the excess length of the socket pins after soldering the board. One of those long pins had bent over and was possibly shorted to another, on the lower left socket. I straightened the pin and re-ran the test. Voila! It passed!

After I finished patting myself on the back I tried to run the spreadsheet again. It still failed! This morning I put the cards back in their usual homes, and everything works fine.

Tuesday Afternoon....Lo and behold, the card is still bad. I found the STB Systems diskette, and ran their RAM test program.

It identified the same chip as being bad. But after running the test for several hours, the errors stopped. Obviously the chip's problems are intermittent.

Wednesday Morning....The chip is still giving errors. I called STB and they said to bring the board by. Wednesday afternoon....STB replaced the chip, and all is well.

```
                  1000 *SAVE S.TEST STB-128
                  1010 *--------------------------------
                  1020 *      TEST STB-128
                  1030 *--------------------------------
0000-             1040 YSAVE  .EQ 0
0001-             1050 LIMIT  .EQ 1
0002-             1060 ADDR   .EQ 2,3
0004-             1070 BANK   .EQ 4
0005-             1080 BYTE   .EQ 5
0006-             1090 SCREEN .EQ 6,7
                  1100 *--------------------------------
C080-             1110 SELECT .EQ $C080
                  1120 *--------------------------------
0800- 20 0D 08    1130 TTTT   JSR TEST
0803- 20 0D 08    1140        JSR TEST
0806- 20 0D 08    1150        JSR TEST
0809- 20 0D 08    1160        JSR TEST
080C- 60          1170        RTS
                  1180 *--------------------------------
080D- A9 00       1190 TEST   LDA #0
080F- 85 04       1200        STA BANK
0811- 85 02       1210        STA ADDR
0813- 20 58 FC    1220        JSR $FC58    CLEAR SCREEN
0816- A9 04       1230        LDA #$04
0818- 85 07       1240        STA SCREEN+1
081A- A9 28       1250        LDA #$28
081C- 85 06       1260        STA SCREEN
                  1270 *---SELECT BANK-----------------
081E- A5 04       1280 .1     LDA BANK
0820- 8D 87 C0    1290        STA SELECT+$07
0823- 09 B0       1300        ORA #$B0     CONVERT TO SCREEN ASCII
0825- A0 00       1310        LDY #0
0827- 91 06       1320        STA (SCREEN),Y
0829- AD 83 C0    1330        LDA SELECT+$03
                  1340 *---TEST D000...DFFF-------------
082C- A9 E0       1350        LDA #$E0
082E- 85 01       1360        STA LIMIT
0830- 20 68 08    1370        JSR TEST.ZEROS
0833- 20 6B 08    1380        JSR TEST.ONES
0836- 20 6E 08    1390        JSR TEST.FIVES
0839- 20 71 08    1400        JSR TEST.AYES
                  1410 *---SWITCH TO OTHER D000---------
083C- AD 8B C0    1420        LDA SELECT+$0B
083F- AD 8B C0    1430        LDA SELECT+$0B
                  1440 *---TEST D000...FFFF-------------
0842- A9 00       1450        LDA #0
0844- 85 01       1460        STA LIMIT
0846- 20 68 08    1470        JSR TEST.ZEROS
0849- 20 6B 08    1480        JSR TEST.ONES
084C- 20 6E 08    1490        JSR TEST.FIVES
084F- 20 71 08    1500        JSR TEST.AYES
                  1510 *---NEXT BANK--------------------
0852- A5 06       1520        LDA SCREEN
0854- 49 80       1530        EOR #$80
0856- 85 06       1540        STA SCREEN
0858- 30 02       1550        BMI .2
085A- E6 07       1560        INC SCREEN+1
085C- E6 04       1570 .2     INC BANK
085E- A5 04       1580        LDA BANK
0860- C9 08       1590        CMP #8
0862- 90 BA       1600        BCC .1
                  1610 *---SWITCH TO ROMS---------------
0864- AD 81 C0    1620        LDA SELECT+$01
0867- 60          1630        RTS
                  1640 *--------------------------------
```

```
                    1650 TEST.ZEROS
0868- A9 00         1660        LDA #0
086A- 2C            1670        .HS 2C       SKIP
                    1680 TEST.ONES
086B- A9 FF         1690        LDA #$FF
086D- 2C            1700        .HS 2C       SKIP
                    1710 TEST.FIVES
086E- A9 55         1720        LDA #$55
0870- 2C            1730        .HS 2C       SKIP
                    1740 TEST.AYES
0871- A9 AA         1750        LDA #$AA
0873- 85 05         1760        STA BYTE
0875- A9 D0         1770        LDA #$D0
0877- 85 03         1780        STA ADDR+1
0879- 20 88 08      1790 .1     JSR FILL
087C- 20 92 08      1800        JSR COMPARE
087F- E6 03         1810        INC ADDR+1
0881- A5 03         1820        LDA ADDR+1
0883- C5 01         1830        CMP LIMIT
0885- D0 F2         1840        BNE .1
0887- 60            1850        RTS
                    1860 *-------------------------------
0888- A0 00         1870 FILL   LDY #0
088A- A5 05         1880        LDA BYTE
088C- 91 02         1890 .1     STA (ADDR),Y
088E- C8            1900        INY
088F- D0 FB         1910        BNE .1
0891- 60            1920        RTS
                    1930 *-------------------------------
                    1940 COMPARE
0892- A0 00         1950        LDY #0
0894- B1 02         1960 .1     LDA (ADDR),Y
0896- C5 05         1970        CMP BYTE
0898- D0 04         1980        BNE .3
089A- C8            1990 .2     INY
089B- D0 F7         2000        BNE .1
089D- 60            2010        RTS
089E- 48            2020 .3     PHA          SAVE ACTUAL DATA
089F- 84 00         2030        STY YSAVE    SAVE Y-REG
08A1- A5 03         2040        LDA ADDR+1   PRINT ADDRESS OF FAILURE
08A3- A0 02         2050        LDY #2
08A5- 20 CA 08      2060        JSR CONBYTE
08A8- A5 00         2070        LDA YSAVE    LO-BYTE OF ADDRESS
08AA- 20 CA 08      2080        JSR CONBYTE
08AD- C8            2090        INY
08AE- A5 05         2100        LDA BYTE     WHAT DATA SHOULD HAVE BEEN
08B0- 20 CA 08      2110        JSR CONBYTE
08B3- C8            2120        INY
08B4- 68            2130        PLA          WHAT DATA REALLY WAS
08B5- 48            2140        PHA          KEEP ON STACK TOO
08B6- 20 CA 08      2150        JSR CONBYTE
08B9- C8            2160        INY
08BA- 68            2170        PLA          FIGURE WHICH BITS WERE WRONG
08BB- 45 05         2180        EOR BYTE
08BD- 20 CA 08      2190        JSR CONBYTE
08C0- A0 00         2200        LDY #0       DELAY LOOP TO SLOW THINGS DOWN
08C2- 88            2210 .4     DEY          FOR OBSERVATION
08C3- D0 FD         2220        BNE .4
08C5- A4 00         2230        LDY YSAVE
08C7- 4C 9A 08      2240        JMP .2       REJOIN TEST
                    2250 *-------------------------------
                    2260 CONBYTE
08CA- 48            2270        PHA
08CB- 4A            2280        LSR
08CC- 4A            2290        LSR
08CD- 4A            2300        LSR
08CE- 4A            2310        LSR
08CF- 20 D3 08      2320        JSR CONNYBBLE
08D2- 68            2330        PLA
                    2340 CONNYBBLE
08D3- 29 0F         2350        AND #$0F
08D5- C9 0A         2360        CMP #10
08D7- 90 02         2370        BCC .1
08D9- 69 06         2380        ADC #6
08DB- 69 B0         2390 .1     ADC #$B0
08DD- 91 06         2400        STA (SCREEN),Y
08DF- C8            2410        INY
08E0- 60            2420        RTS
                    2430 *-------------------------------
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

Single-Step *mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

Trace *mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

Background *mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

QUICKTRACE *allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

Two optional display formats *can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

QUICKTRACE *is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

QUICKTRACE *is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

QUICKTRACE *is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while* ***QUICKTRACE*** *is alive.*

QUICKTRACE *is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE ***$50***

Is a trademark of Anthro-Digital, Inc.

Copyright © 1981
Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

Procedure for Converting S-C Source Files to Text Files Without Owning an S-C Assembler

............Bob Sander-Cederlof

Strangely enough, there are some of you who still do not own an S-C Assembler. And some of you buy or would like to buy our Quarterly Disks or the Applesoft Docu-Mentor disks.

These disks contain source files which are only usable by the S-C Macro Assembler. However, it is possible (even without an S-C Assembler) to convert them to regular text files so as to be readable by another brand assembler/editor.

The files appear in the catalog as type "I", which is supposed to mean Integer BASIC. Of course the contents has nothing to do with Integer BASIC, but making them "I-files" has several advantages:

* they LOAD/SAVE faster than text files
* standard DOS commands can be used for load/save
* when the S-C Assembler is in the RAM card, DOS can automatically switch between Applesoft and Assembler as it normally would between Applesoft and Integer BASIC.

There are also some dis-advantages:

* some users have trouble believing they are not really Integer BASIC programs, and try to RUN them.
* the files are harder for people without an S-C Assembler to convert to another brand.

Which brings us back to the point of this article.

To make the procedure simple, you need at least a 64K Apple. If you have an Apple //e, you are all set. An older Apple needs a "language card", or "RAM card".

The first step in the conversion process is to load the file into memory and find out where it is. Start by booting with your DOS 3.3 System Master disk, which loads Integer BASIC into the RAM card. Then LOAD the S-C source file which you want to convert. Integer BASIC will be switched on, but don't try to LIST or RUN!

Enter the Monitor by typing "CALL -151". At this point you will get an asterisk prompt. Look at locations $4C, $4D, $CA, and $CB. You can do it like this:

```
*4C.4D CA.CB
004C- 00 96
00CA- 58 73
```

Interpret the above as meaning that the source code begins in memory at $7358 and ends one byte before $9600.

If you use the monitor commands to look at the first 30 or 40 bytes (or more), you will discover how the source lines are stored. Each line begins with a byte count, which if added to the address will give the address of the first byte of the next line. Each line ends with a 00 byte. The byte count includes both of these bytes, and all in between. Here is a sample line:

```
0F E8 03 41 42 43 84 4C 44 41 81 23 24 35 00
```

The second and third bytes are the binary form of the line number. As usual in 6502 domain, the number is stored low-byte first. $3E8 means the line above is line 1000.

The fourth byte and beyond are ASCII codes for the text of the line, with two exceptions. If the bytes are less than $80, they are plain ASCII. If they are in the range from $81 through $BF, they represent a series of blanks. $81 means one blank, $84 means four blanks, and so on. The line above now decodes to:

```
1000 ABC    LDA #$5
```

The other exception is not illustrated above, but here is one:

```
08 F2 03 2A C0 20 2D 00
```

The token $C0 means "repeated character". The next byte after $C0 gives the number of repetitions, and the byte after that tells what character to repeat. Above the C0 20 2D means 32 "-" characters, so the whole line looks like this:

```
1010 *--------------------------------
```

Armed with all that information, you can probably see how to write a simple Applesoft program to convert the memory image of the S-C source file to plain text and then write it on a text file.

In fact, here is just such a program:

```
100  REM CONVERT MEMORY IMAGE OF S-C SOURCE
110  REM TO ORDINARY TEXT FILE
200 HM =  PEEK (76) + 256 *  PEEK (77)
210 PP =  PEEK (768) + 256 *  PEEK (769)
220  HIMEM: PP
300  REM OPEN A TEXT FILE
310 D$ =  CHR$ (4)
320  PRINT D$"OPEN TEXTFILENAME": PRINT D$"DELETE TEXTFILENAME"
330  PRINT D$"OPEN TEXTFILENAME": PRINT D$"WRITE TEXTFILENAME"
400 L = PP
410  IF L = HM THEN  PRINT D$"CLOSE": END
420  GOSUB 500: REM  DO ONE LINE
430  GOTO 410
500  REM DO ONE SOURCE LINE
510 N =  PEEK (L)
520 LN =  PEEK (L + 1) + 256 *  PEEK (L + 2): PRINT LN" ";:L = L + 2
530 L = L + 1:C =  PEEK (L): IF C = 0 THEN  PRINT :L = L + 1: RETURN
540  IF C < 128 THEN  PRINT  CHR$ (C);: GOTO 530
550  IF C < 192 THEN  FOR I = 1 TO C - 128: PRINT " ";: NEXT : GOTO 530
560  IF C = 192 THEN  FOR I = 1 TO  PEEK (L + 1): PRINT  CHR$ ( PEEK (L +
     2));: NEXT I:L = L + 2: GOTO 530
570  PRINT : PRINT D$"CLOSE": PRINT "***ERROR IN SOURCE AT "L"****": END
```

Here is a blow-by-blow description of how to use the program.

1. Boot your DOS System Master to load INTBASIC into the RAM card.
2. Load the S-C source file.
3. Type CALL-151 to get into the monitor.
4. Type CA.CB to get the starting address of the S-C source program (xx yy).
5. Type 300:xx yy to store the starting address in a place Applesoft will not clobber.
6. Type 3D0G to return to Integer BASIC.
7. Type RUN CONVERT S-C TO TEXT to execute the Applesoft program listed above.
8. Stand back and wait while the program chugs through the bytes. When you see the Applesoft prompt again, it is all done!

If you add a line at 315 to turn on MONCIO, you can see the text as it is produced.

Where To?, Revisited................................Bill Morgan

Many thanks to all of you who responded to my questions abcut 68000, C, and the future of Apple Assembly Line.

Your answers ran about eight to one in favor of including 68000 information in AAL. Several writers suggested starting with a few pages, and possibly splitting off a separate newletter someday. That sounds like a good plan, so we'll start a regular section next issue. Those of you who already know 68000 can now start teaching the rest of us. Bob Urschel has already sent in a brief article and program! He has a QWERTY Q68 board like that we reviewed last month, and speaks very highly of it.

Interest in Mackintosh (MacIntosh? Apple 32?) is growing rapidly: the announcement is expected at the Apple shareholder meeting in mid-January. Some reports claim that some developers have had Mac for up to 18 months now. We haven't been among those so privileged, but I hope to be the first on my block with one. (Unless the thing turns out to have some fatal flaw, like no expansion slots. That was one rumor!)

Several of you also expressed an interest in C, but not even a majority. More like 30%. It looks like a number of people are curious, but feel that too much coverage would dilute AAL. Stephen Bach said it best, "... don't spread yourselves too thin and try to do C also." I expect to do occasional reviews and mentions of books and other aids to learning C, and to report on anything specifically related to C on Apple computers, but not much more.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).